Universität Bielefeld – Technische Fakultät
AG Rechnernetze und verteilte Systeme

Vorlesung 2

Rechnerarchitektur

Wintersemester 2001/2002

Peter B. Ladkin
ladkin@rvs.uni-bielefeld.de

Inhalt

- Allgemeine von Neumann Architektur
- CPU Architektur
- Wie sie funktionieren
- Assembly Sprache
- Befehl Ausführung

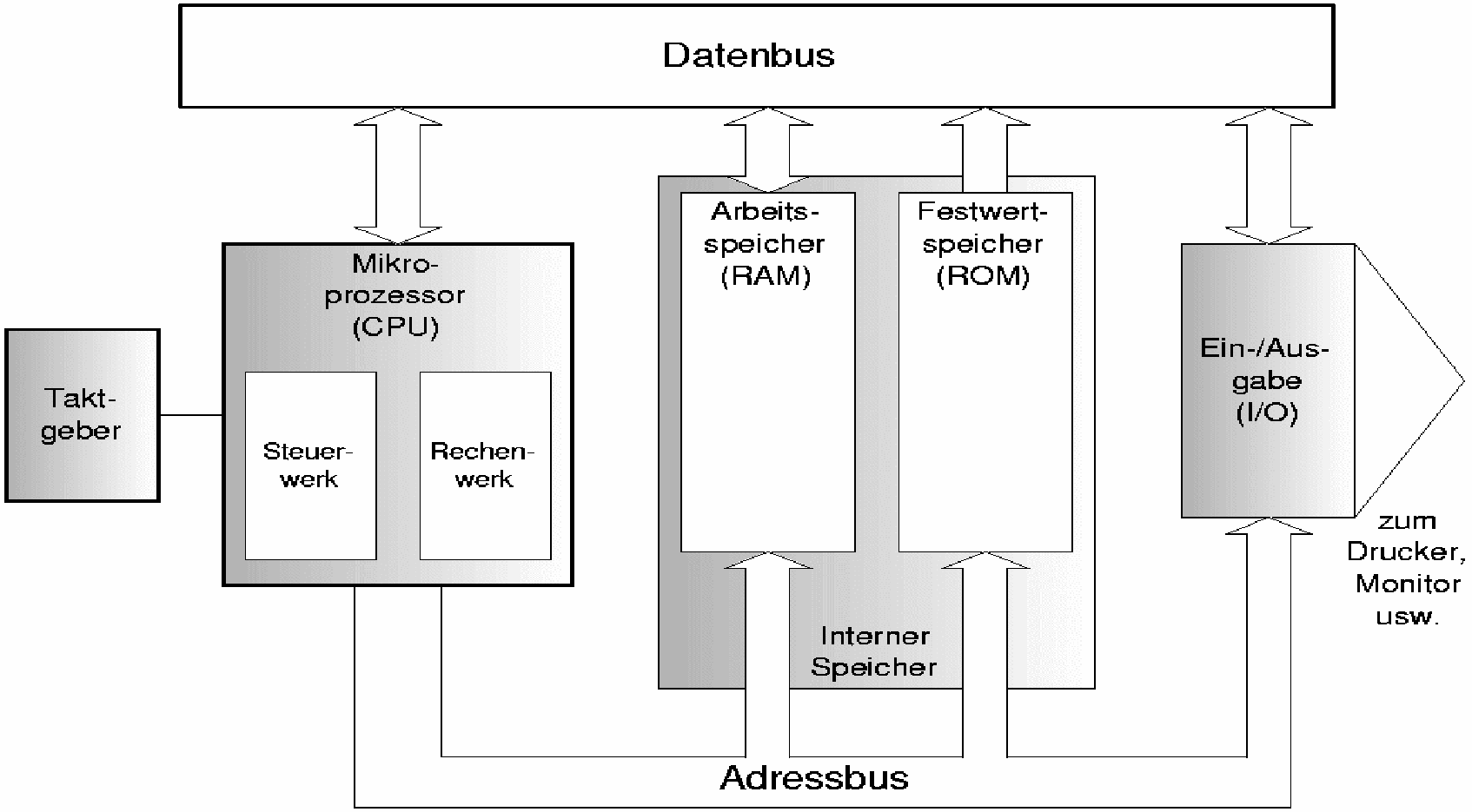
Von Neumann Architektur

- Stored Program Architektur
- Speicher beinhaltet Daten
- Speicher beinhaltet auch das Programm
- Befehle werden vom Speicher geholt
- Sowie Daten
- Speicher in Daten/Programm Speicher geteilt

Vorher

- Programme per Laufbandleser gelesen
- "Paper Tape" - Papierband mit Löcher
- Oder per "punched card". Hollerith (IBM)
- Daten auch
- Aber Zwischenresultate gespeichert

Allgemeine vN-Achitektur



Adressen

- Speicher ist ein Array von erheblicher Grösse
- Adressen sind wie Array-Indices
- Die Speicherhardware "interpretiert" eine Adresse
- Adressen laufen den Speicher hin auf dem Adressbus

Daten

- Daten sind der Inhalt eines Array-Elementes
- Wenn eine Adresse "hereinkommt", wird der Datum, der in der Adresse gespeichert wird, auf dem Datenbus gesetzt

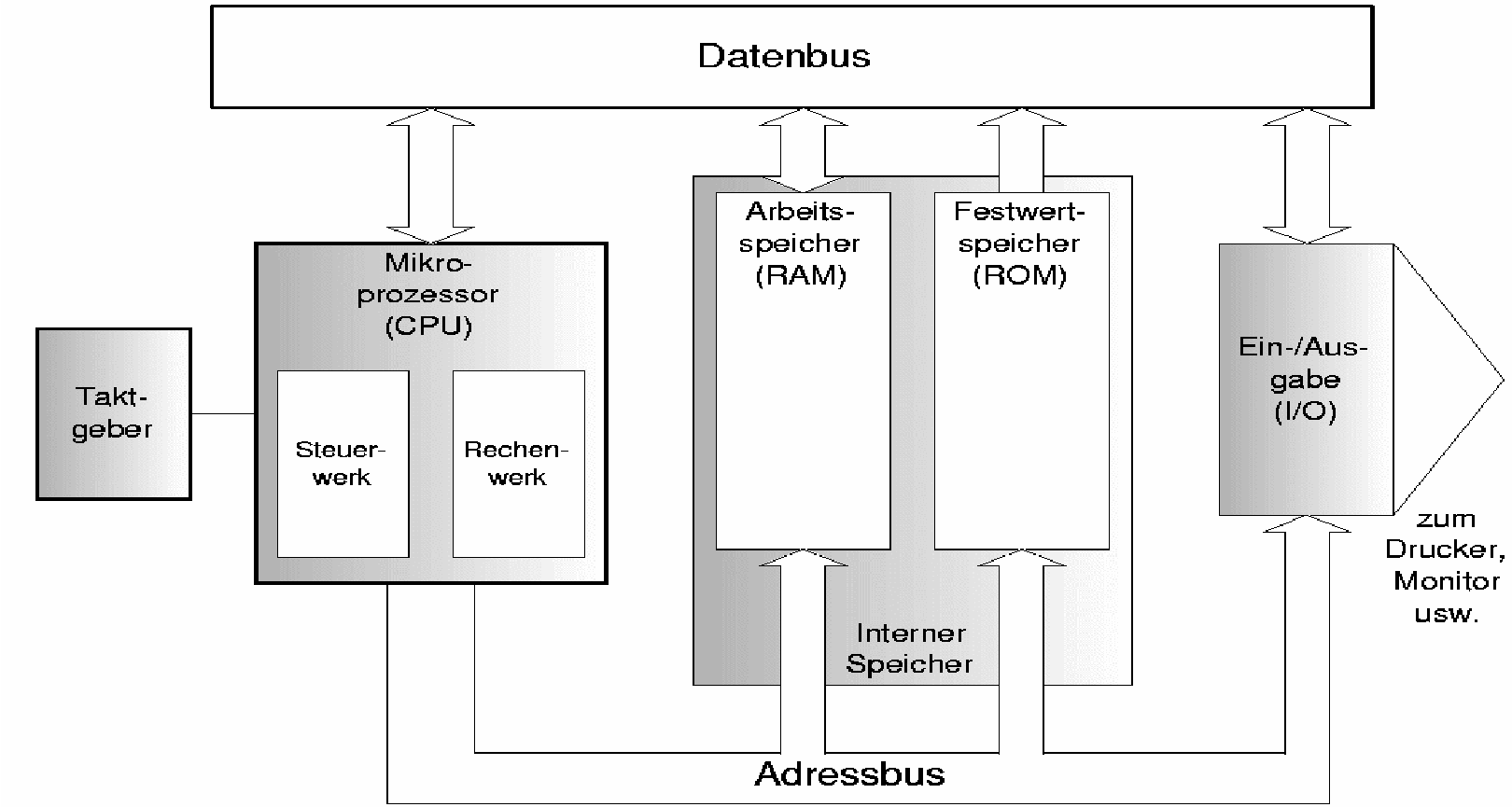
Bits und Strom

- Wie "laufen" Daten und Adressen auf ein Bus?
- Sagen wir mal, die Maschine ist "8-bit"
- Und die Adressen sind auch "8-bit"
- Ein Bus besteht auf 8 parallele Leitungen
- Für ein "1", wird die entsprechende Leitung "hoch" gesetzt (die Spannung wird hoch)
- Für ein "0", auf null gesetzt (Spannung null)
- Für eine entsprechende Zeit

Bits und Strom

- Dies ist nicht die einzige Methode, aber allgemein
- "Zeit" kommt von einer Uhr
- Z.B. die selbe Uhr, die die Taktfrequenz gibt

Architektur nochmal



Wie es funktioniert

- Tick: Adresse auf Adresse-Bus (entsprechende Leitungen hochgesetzt)
- Tick: erreicht Speicher
- Tick, Tick: Speicherelektronik öffnet Adresse
- Tick: Daten auf Datenbus eingeschaltet
- Tick: Datenwelle erreicht CPU
- Tick: CPU schaltet Daten auf ein Register ein

Wie es funktioniert

- Daten laufen auf dem Bus nicht unbedingt zu Takt
- Speicher reagiert nicht undedingt auf Takt
- Aber es gibt eine Anzahl von Ticks in dem man auf jeden Fall die Daten bekommt
- Diese wird "Memory latency" benannt und für alle Speicher-Datenanschlussoperationen benutzt

Wie es funktioniert

- Die meisten CPUs haben internen Speicher, sogenannte "Register"
- Wenn mehrere davon, heisst es "Cache"
- Register sowie Cache reagieren auf Taktfrequenz
- Dann muss Speicher nicht mehr "synchronous" mit dem CPU arbeitet
- Er könnte "asynchronous" mit dem CPU arbeiten

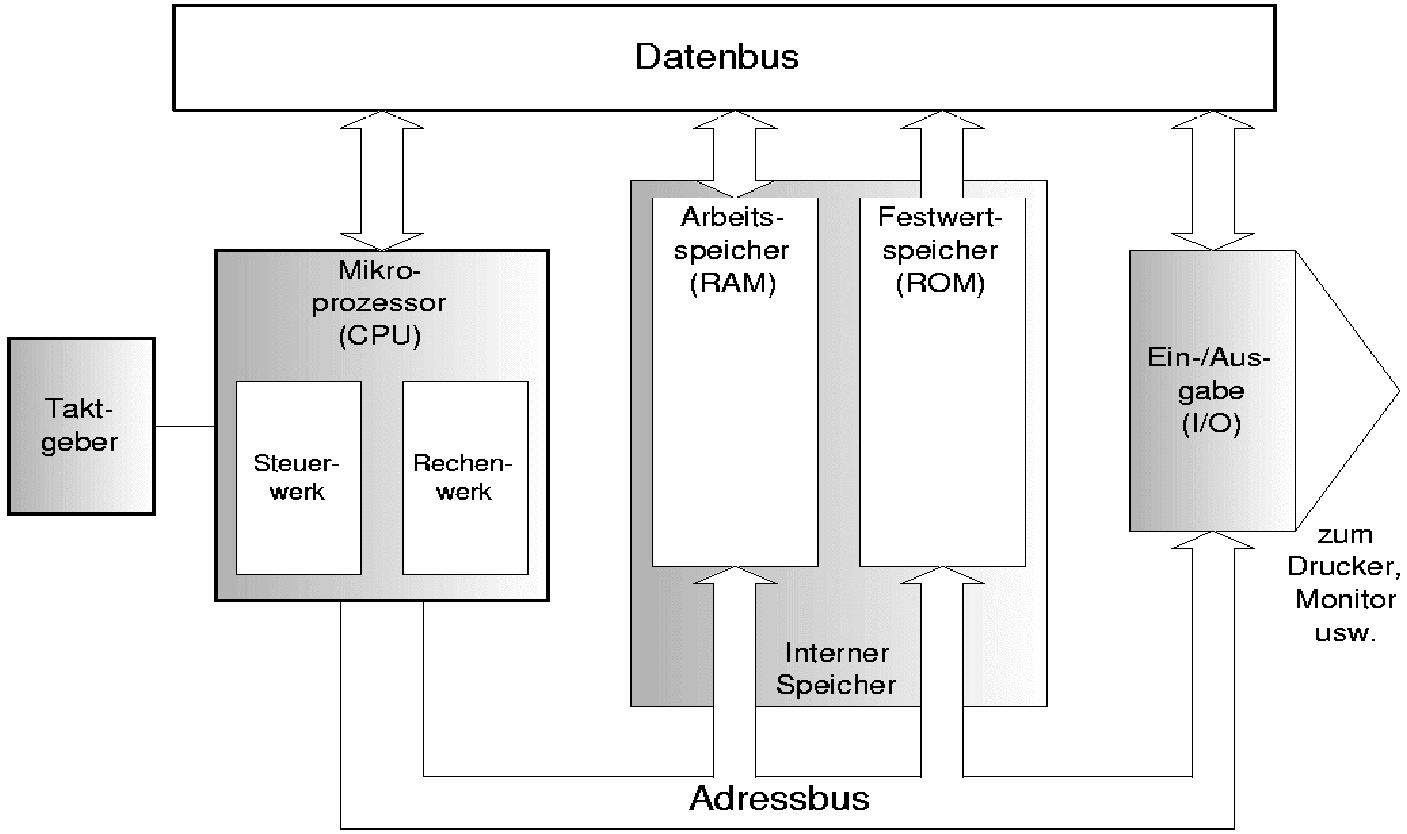
Wie es funktioniert

- Es gibt eine Menge klügere Algorithmen, die Cache und Speicher effizient benutzten
- Die Algorithmen werden in Hochleistungsprozessoren verwendet

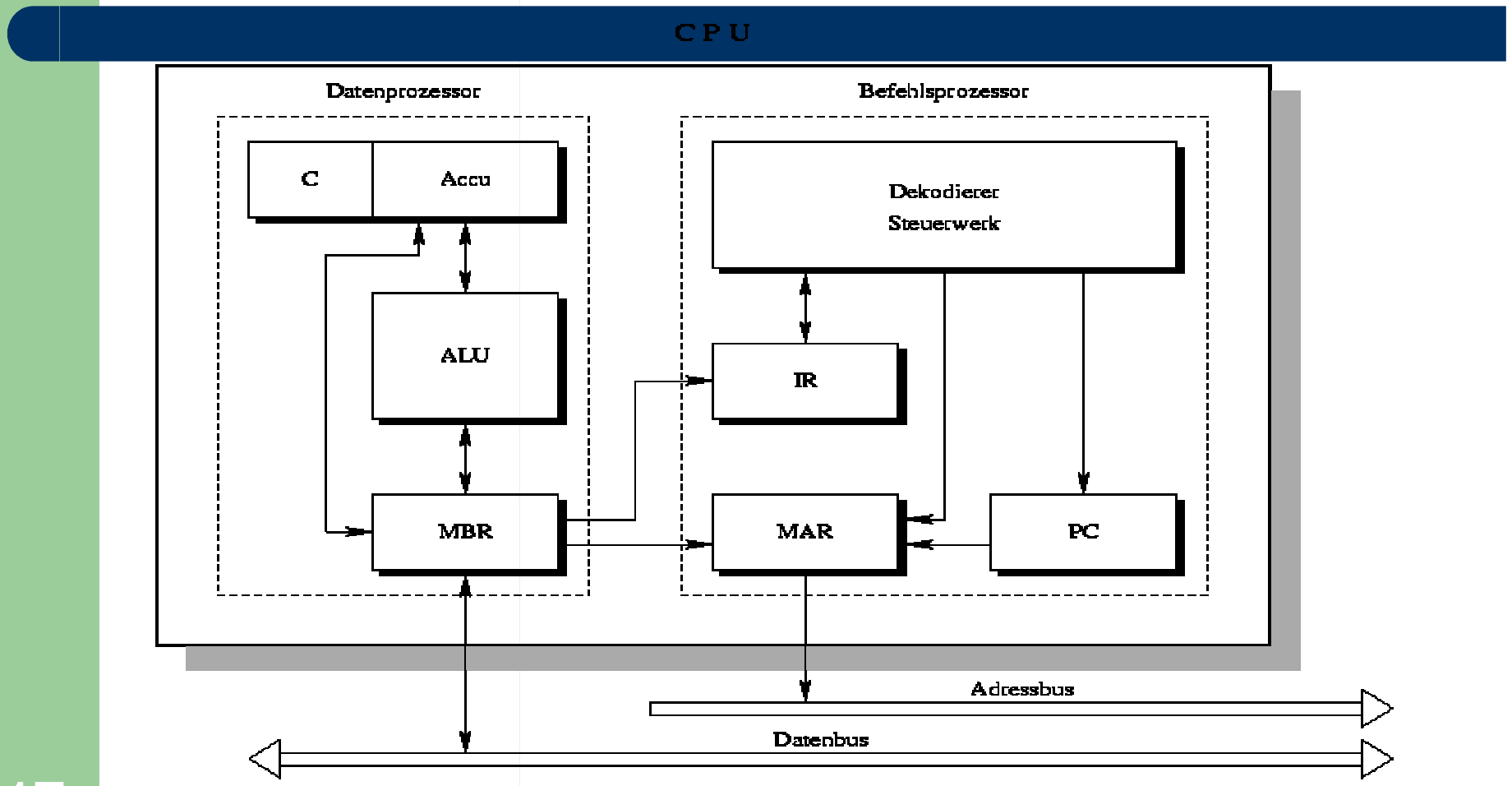
I/O

- Funktioniert wirklich asyncron mit dem CPU
- Und sehr langsam
- Von Software bedient; "Drivers"
- Adressen von I/O-Geräte alle höhe als Adressen vom Speicher

CPU


Datenbus
Mikro-
prozessor
(CPU)
Takt-
geber
Steuer-
werk
Rechen-
werk
Arbeits-
speicher
(RAM)
Festwert-
speicher
(ROM)
Interner
Speicher
Ein-/Aus-
gabe
(I/O)
zum
Drucker,
Monitor
usw.
Adressbus

CPU


CPU
Datenprozessor
Befehlsprozessor
C
Accu
ALU
MBR
Dekodierer
Steuerwerk
IR
MAR
PC
Adressbus
Datenbus

CPU

- PC hält die Adresse des nächsten Befehls
- Tick: wird in den MAR gespeichert
- Tick: wird auf den Adressbus gestellt; PC gleichzeitig incrementiert
- Tick...tick: Daten (Befehl) kommt in MBR ein
- Tick: Befehl wird in IR transferiert
- Tick: Befehl wird in Dekodierer transferiert
- Wird in Befehl und Argumente geteilt

CPU: "Jump"-Befehl

- Falls ein "Jump <Befehl-Adresse>", wird in "Jump" und "<Befehl-Adresse>" geteilt
- <Befehl-Adresse> wird in MAR gesetzt
- Tick...tick neues Befehl kommt in MBR ein
- Wird in IR transferiert
- Fangen wir neu an

CPU: "Add"-Befehl

- Falls ein "Add"-Befehl
- "Add" wird an ALU weitergegeben
- "Add <Speicher-Adresse>" bedeutet:
  - ACC <- <Sp.Ad.-Inhalt> + ACC
- Speicher-Adressen werden MAR -> MBR Zyklus folgen

CPU: "Add"-Befehl

- Tick...tick; Inhalt der <Sp.-Adresse> wird in MBR erscheinen
- ALU transferiert MBR-Inhalt in den ALU
- Diese Inhalt wird am Inhalt des ACCs addiert innerhalb des ALU
- Resultat wird in den ACC transferiert
- PC wird in MAR transferiert usw

Assembly-Sprache

- Wird "Jump xxyyzz" geschrieben statt in binäre Notation
- Ein "Assembler" convertiert die Programme in binäre Notation
- Die binäre Notation ist die Maschinen-Sprache
- Maschinen-Sprache wird von der Maschine ausgeführt

Assembly-Sprache

- Arithmetische Operationen
  - Add: ACC <- ACC + <Sp.Ad.-Inhalt>
  - Subtract: ACC <- ACC - <Sp.Ad.-Inhalt>
  - Shift (multiply by 2): ACC <- ACC * 2 == alle Bits in ACC einmal nach links gestellt mit 0 in Low-Order Bit
  - Multiply: ACC <- ACC * <Sp.Ad.-Inhalt>
  - Divide: ACC <- ACC / <Sp.Ad.-Inhalt>

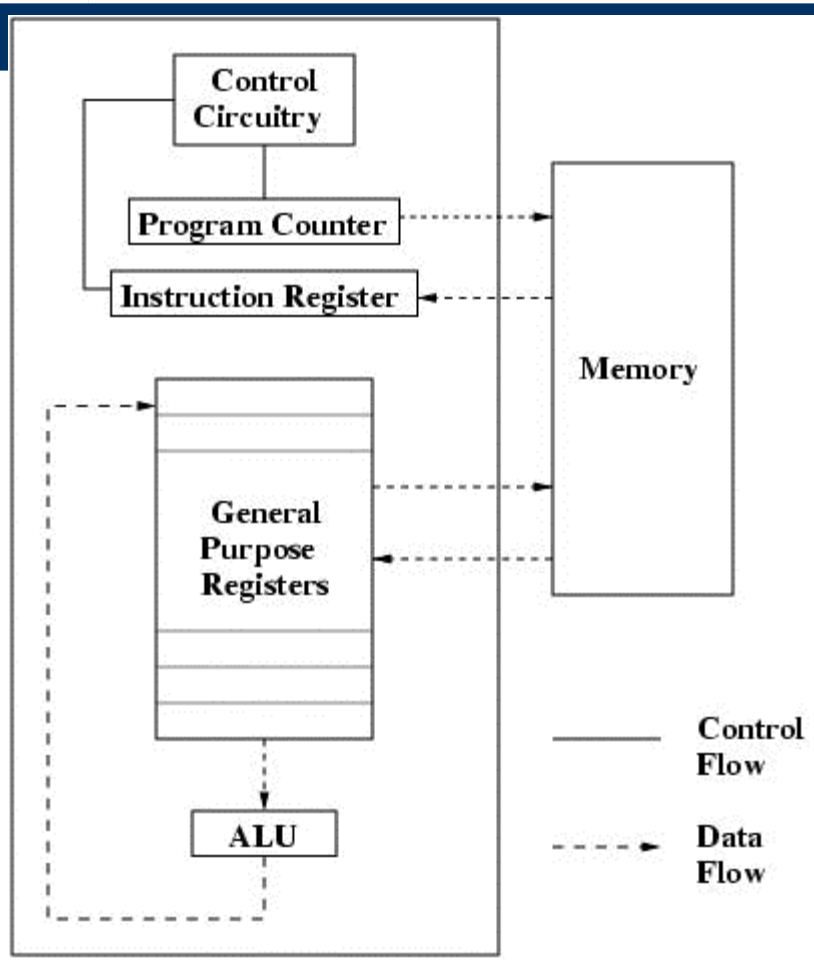
Assembly-Sprache

- Logische Operationen
  - "Jump <Befehl-Ad.>" : Inhalt der <Bef.-Ad.> wird in IR geladen; PC wird <Bef.-Ad.> + 1
  - "Load <Sp.-Ad.>": Inhalt der <Sp.-Ad.> wird in ACC geladen; PC wird PC + 1
  - "Store <Sp.-Ad.>": Inhalt des ACCs wird in <Sp.Ad.> gespeichert; PC wird PC + 1
  - "If ACC then <Bef.-Ad.>": ACC > 0 then PC wird <Bef.-Ad.>; ACC </= 0 then No-Op.

Übung

- Nimm die 9 Befehle
- Beschreib genau, was auf jedem Tick passieren muss, um das Befehl endgültig auszuführen
- Zähl die Ticks pro Befehl
- Stell zusätzliche "null"-Ops herein, um die Anzahl der Ticks für jedes Befehl auszugleichen

Ein abstrakterer CPU

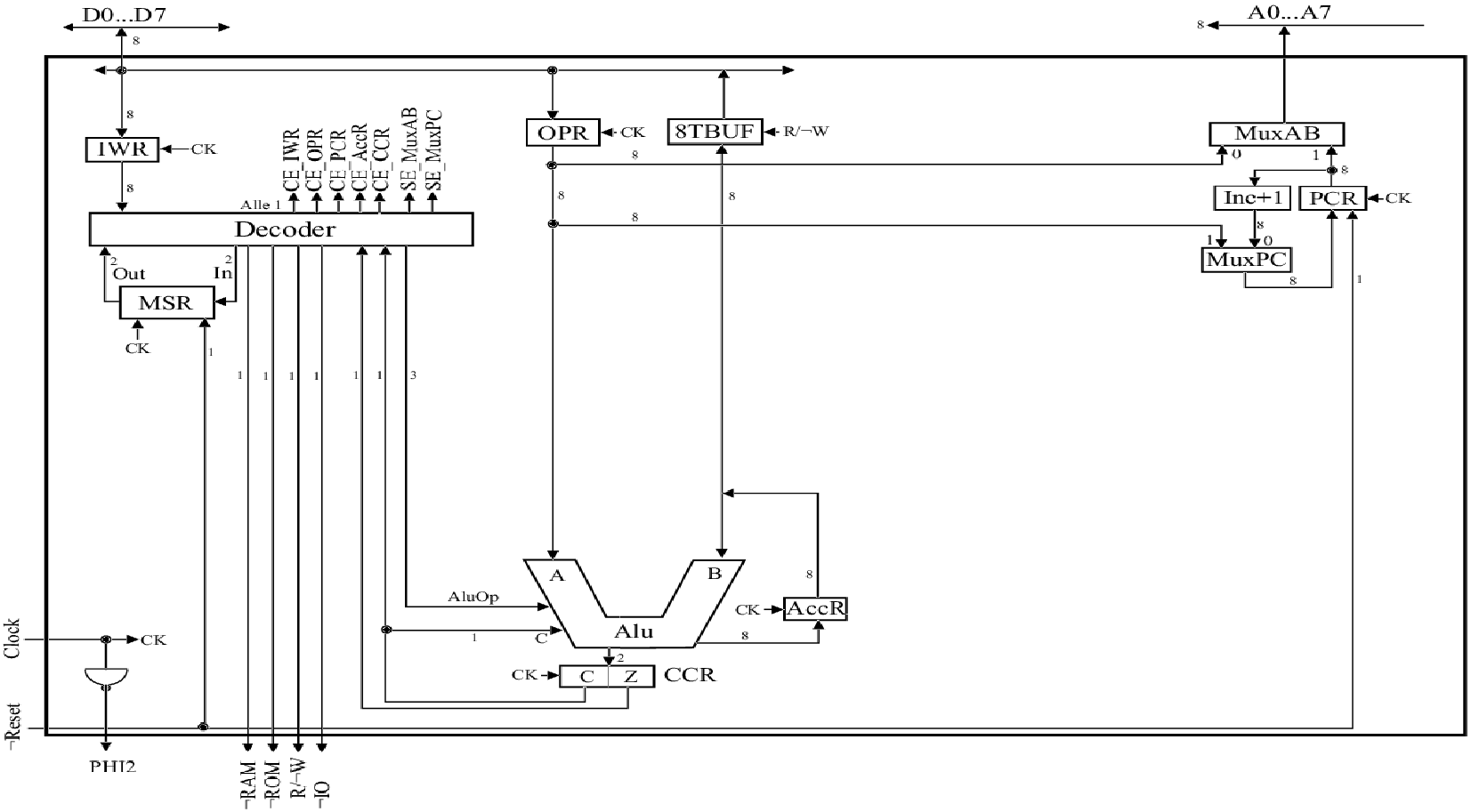


CPU

CPU - vermisste Teile

- Interrupt-Werk wird nicht bezeichnet
- Genau wie die Uhr funkioniert, wird auch nicht bezeichnet

CPU des DEPs


D0...D7
A0...A7
IWR
CK
CE_IWR
CE_OPR
CE_PCR
CE_AccR
CE_CCR
SE_MuxAB
SE_MuxPC
Alle 1
Decoder
OPR
8TBUF
R/¬W
MuxAB
Inc+1
PCR
MuxPC
Out
In
MSR
A
B
AluOp
Alu
C
AccR
CCR
Z
Clock
¬Reset
PHI2
¬RAM
¬ROM
R/¬W
¬IO